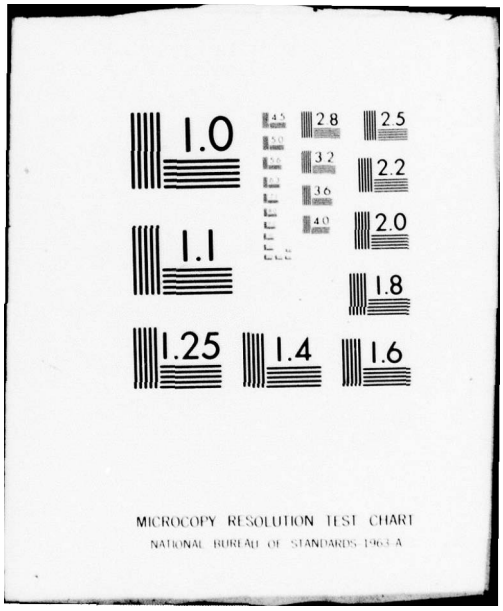
AD-A068 967 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 5/7
SPECIFYING THE SEMANTICS OF WHILE-PROGRAMS: A TUTORIAL AND CRIT--ETC(U)
MAR 79 I GRIEF, A R MEYER N00014-75-C-0661
UNCLASSIFIED MIT/LCS/TM-130 NL

| OF |
AD A068967

END
DATE FILMED
6-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-130

SPECIFYING THE SEMANTICS OF *WHILE*-PROGRAMS:

A Tutorial and Critique of a Paper by Hoare and Lauer

Irene Greif

Albert R. Meyer

April 1979

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0661, and in part by the National Science Foundation under grants MCS76-17998 and MCS77-18754 A03

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TM-130
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Specifying the Semantics of While-Programs: A Tutorial and Critique of a Paper by Hoare and Lauer

5. TYPE OF REPORT & PERIOD COVERED:

6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TM-130

7. AUTHOR(s): Irene Greif and Albert R. Meyer

8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0661, NSF-MCS78-17698, MCS77-19754 A03

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
MIT/Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
ARPA/Dept. of Defense
1400 Wilson Boulevard
Arlington, VA 22209

NSF/Associate Program Dir.
Office Computing Activities
Washington, D. C. 20550

12. REPORT DATE: March 1979

13. NUMBER OF PAGES: 38

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
ONR/Dept. of the Navy
Information Systems Program
Arlington, VA 22217

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
semantics of programming languages

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

We consider three kinds of mathematical objects which can be designated as the "meaning" or "semantics" of programs: binary relations between initial and final states, binary relations on predicates (partial correctness semantics), and functionals from predicates to predicates (predicate transformers). We exhibit various formal specification mechanisms: induction on program syntax, axioms, and deductive systems. We show that each kind of semantics can be specified by several different mechanisms. As long as arbitrary predicates on states are permitted, each kind of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 648

semantics uniquely determines the others -- with the sole exception of the weakest pre-condition semantics for nondeterministic programs.

MIT/LCS/TM-130

SPECIFYING THE SEMANTICS OF WHILE-PROGRAMS:

A Tutorial and Critique of a Paper by Hoare and Lauer

Irene Greif and Albert R. Meyer

March 1979

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract No. N00014-75-C-0661, and in part by the National Science Foundation under grants MCS78-17698 and MCS77-19754 A03.

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge Massachusetts 02139

SPECIFYING THE SEMANTICS OF *WHILE*-PROGRAMS:

A Tutorial and Critique of a Paper by Hoare and Lauer

by

Irene Greif and Albert R. Meyer

Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

We consider three kinds of mathematical objects which can be designated as the "meaning" or "semantics" of programs: binary relations between initial and final states, binary relations on predicates (partial correctness semantics), and functionals from predicates to predicates (predicate transformers). We exhibit various formal specification mechanisms: induction on program syntax, axioms, and deductive systems. We show that each kind of semantics can be specified by several different mechanisms. As long as arbitrary predicates on states are permitted, each kind of semantics uniquely determines the others -- with the sole exception of the weakest pre-condition semantics for nondeterministic programs.

KEY WORDS: semantics of programming languages

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0661, and in part by the National Science Foundation under grants MCS78-17698 and MCS77-19754 A03.

1. Introduction

Our aim in this paper is to clarify the characteristics of a proper specification of programming language semantics. We illustrate alternative specifications of several different kinds of semantical objects and examine the extent to which these different semantics capture the same information about programs.

Hoare and Lauer [1974] have advocated using a variety of styles of programming language definitions to fit the variety of users from implementers to program verifiers. They consider the question of whether different specifications determine the same language by showing that the specifications are what they call "consistent". However, their treatment skirts the question of whether the specifications can each be taken to determine the language adequately.[1] Although, as we will show, any one of the kinds of semantical specifications they discuss -- operational definitions, relational "theories," and partial correctness assertions -- can be used to determine meaning uniquely, Hoare and Lauer do not make the case in their paper. In fact, both their relational and partial correctness specifications are satisfied by several different semantics, only one of which is desired.

We basically agree with Hoare and Lauer that alternate specifications can and should be given, but feel that the difficulties noted above indicate the need for more careful attention. Additional pitfalls which we attempt to avoid include confusion between the mathematical object which is designated to be the meaning of a program and *methods for specifying that object*; confusion between consistency and equivalence of two definitions; between completeness of a theory and its having a unique model. While these issues are familiar in mathematical logic, we take this opportunity to survey them in the context of the programming language semantics of a trivial class of *while*-programs. Because these programs are trivial, none of the challenging research problems concerned with explaining how complex programs behave, or what they "mean," can arise. This allows us to focus more clearly on the way in which the semantics are specified, without being distracted by any difficulty in understanding what that semantics may be.

Of particular interest to us is the thesis that a programming language semantics can be specified by giving all the "before-after" assertions true of programs in the language. This thesis appears first to have been put forward by the title of Floyd's seminal paper [1967]. Hoare and Wirth [1973] carried out the first serious attempt to apply the thesis in practice by specifying the semantics of a substantial fragment of the programming language PASCAL in this way. More recently, Dijkstra has advocated a similar approach to explaining semantics [Dijkstra, 1976, p. 17]:

> "...we know the possible performance of the ... [program α] ... sufficiently well, provided that *we can derive for any post-condition* ... [Q] ... the corresponding weakest pre-condition ... [$wp_\alpha(Q)$] ... because then we have captured what the ... [program] ... can do for us; and in the jargon the latter is called 'its semantics'."

In Sections 2 and 3 we consider different techniques for specifying the input-output behavior, i.e., relational semantics, of programs. In Sections 4 and 5 we analyze semantics based on sets of partial correctness assertions and weakest pre-conditions.

Since most of the proofs are entirely routine, we have postponed them to appendices. Nevertheless, for completeness most of the proofs are given.

This paper can be read without prior familiarity with Hoare and Lauer's paper.

2. The Programming Language and Meanings for It

2.1 *While*-Programs

Following Hoare and Lauer, we examine alternative specifications of the meaning of a trivial programming language with primitive statements, *while* statements, and statement lists. The syntax, omitting details of the form of predicate expressions is as follows:

<program> ::= <primitive statement> | <while statement> |
<program>;<program> | *NOP*

<while statement> ::= *while* <predicate expression> *do* <program>

As is usual with abstract syntax, we will not concern ourselves with ambiguity in parsing or with detailed syntax of expressions and primitive statements.

We assume that programs run on machines with states. We treat the states simply as abstract elements in some fixed set S, ignoring their internal structure. In many familiar examples primitive statements define total functions from states to states, but we need not make this assumption. Primitive statements may be *partial*, i.e. for some state s there may be no related state, and *nonfunctional*[2], i.e. for some states s there may be more than one related state. A primitive statement, A, thus has an effect on states which can be defined by giving an *initial-state, final-state relation* $R_A \subset S \times S$ such that $(s, s') \in R_A$ *iff* A executed in s can terminate in state s'.

Example: Suppose that a state is an assignment of values to variables (to be thought of as a state of computer memory giving the contents of all the registers, arrays, etc.), and consider the primitive assignment statement *choose* u *in* U where u is a variable of some basic type and U is a variable ranging over finite sets of elements of the same basic type. Then $(s,s') \in R_{\textit{choose } u \textit{ in } U}$ *iff* $s(X) = s'(X)$ for all variables $X \neq u$ and $s'(u) \in s(U)$.

Note that $R_{\textit{choose } u \textit{ in } U}$ is partial because $s(U)$ may be empty, and is nonfunctional because $s(U)$ may have more than one element.∎

A predicate P is a mapping from states to truth values. Predicate expressions p, q,... appear in programs. We will use P, Q,..., respectively, to denote the predicates corresponding to these expressions.[3] For simplicity, we assume that predicate expressions always yield values, so that the predicate P associated with an expression p is true or false at each state and is never undefined.

We use the following notation throughout what follows:

a, b, c	programs,
A	primitive statements,
s, t	states (elements of the set S of all states),
p, q	predicate expressions,
P, Q	predicates on states,
L, M, R	binary relations on states (subsets of S×S),
$\mathcal{M}, \mathcal{R}$	binary relations on predicates (subsets of $2^S \times 2^S$).

Each of these letters may appear with subscripts or multiply primed, e.g., s_1, s', Q", etc.

2.2 Semantics and Specifications

A *semantics* for a programming language is a mapping from programs to objects in a domain of *meanings*. Examples of meanings are sets of state sequences, relations on states, relations on predicates, and functionals on predicates (predicate transformers). A semantical *specification* determines such a mapping, or perhaps a family of acceptable mappings, from programs to meanings. Thus, we do not require that a specification determine semantics uniquely. For example, in a typical specification error messages may be left to an implementer's discretion. Differing implementations of error messages will then correspond to differing semantical mappings which satisfy the specification (*cf.* section 4.5).

A semantics is a mathematical object, and distinctions among semantics can be made precisely. In contrast, our classification of specification techniques is informal -- we do not attempt to give a precise or exhaustive characterization of methods for specifying semantics. Examples of specification techniques include operational definitions, inductive definitions, axioms, and deductive systems. It should be clear by example below what we will mean by the last three. Loosely, what we mean by an operational definition is one which has a computational flavor reflecting the step by step execution of programs. Operational definitions are not considered in this paper. Two examples appear in Hoare and Lauer's paper in their "interpretive" and "computational" definitions. In particular, their interpretive definition is an abstract machine which can execute program steps.

A semantics can also have a computational flavor. An example is a mapping of programs into sets of state sequences, where each sequence consists of the successive states which are reached during execution of the program. The second "computational" definition in the Hoare and Lauer paper is of just this semantics.

We note that such an "operational" semantics need not be *specified* operationally. In fact, we can easily write a concise inductive specification of this semantics for our language. Each program a is mapped into the set of state sequences (or trajectories) Tr_a. We assume that the initial-state, final-state relations, R_A, of primitive programs A are given. The state sequence meaning, Tr_A, of A is the set of sequences ss' such that $(s,s') \in R_A$. The state sequence meanings of other programs are defined by induction on program structure, as follows:

TR1. $Tr_{NOP} = \{ss \mid s \text{ a state}\}$,

TR2. $Tr_{a;b} = Tr_a : Tr_b$,

TR3. $Tr_{\textit{while}\ p\ \textit{do}\ a} = (Tr_p : Tr_a)^{:*} : Tr_{\neg p}$.

where $Tr_a : Tr_b$ is the set of state sequences $s_1 \ldots s_2 \ldots s_3$ such that $s_1 \ldots s_2 \in Tr_a$ and $s_2 \ldots s_3 \in Tr_b$, $Tr_p = \{ ss \mid P(s)\}$, and $(Tr)^{:*} = \Lambda \cup Tr \cup Tr:Tr \cup \ldots$, where Λ is the singleton set containing the *null* sequence (defined to act as an identity element under the ":" operation).[4]

This association of sets of state sequences to programs is the same as that specified by Hoare and Lauer, and it clearly describes a reasonable kind of semantics for our language. To see this, note that *NOP* does not change the state, and the program $a;b$ will follow a computation sequence $s \ldots s'$ *iff* the program a started in s can follow a computation sequence $s \ldots t$ and the program b can follow $t \ldots s'$. Similarly, for *while* loops, if $P(s)$ then $s \ldots s'$ is a computation sequence *iff* there are sequences $s \ldots s_1$, $s_1 \ldots s_2$, ..., $s_k \ldots s'$ which are computations of a, $P(s_i)$ for all the s_i, $\neg P(s')$, and $s \ldots s' = s \ldots s_1\, s_1 \ldots s_2 \ldots s_k \ldots s's'$. If $\neg P(s)$ then *while* p *do* a acts like a *NOP* on s, that is, $ss \in Tr_{\textit{while}\ p\ \textit{do}\ a}$.

Hoare and Lauer show that their two operational definitions are "consistent" in that both define the same *standard relational semantics*, **R**, mapping program a to relation R_a. These initial-state, final-state relations can be defined as follows:

$$R_a = \{(s,t) \mid \text{there is a sequence } s \ldots t \in Tr_a\}.$$

Thus the standard relational semantics can be defined in terms of the state sequence semantics. (Consequently any specification of the latter semantics indirectly also specifies the former. It will accordingly be important to keep track of which kind of meanings are being specified in any given context. We illustrate this point further in Section 4.4.)

In addition to considering how a specification determines a semantics, we will examine ways in which differing semantics can determine one another. We shall say that one semantical mapping *determines* another *iff* any two programs which are assigned the same meaning in the meaning domain of the first semantics are also assigned the same meaning in the meaning domain of the second semantics. Two semantics are *equivalent iff* each determines the other. Thus, even if the domains of meaning of two semantics consist of distinct kinds of mathematical objects, it may be that the two semantics can be considered equivalent by this definition.

Equivalent semantics make exactly the same distinctions among programs. Thus the

meaning of a program according to one semantics uniquely determines the meaning of that program according to all equivalent semantics. From a mathematical point of view this means that equivalent semantics provide exactly the same information about programs. Of course the method for transforming one meaning into another may be laborious or otherwise inconvenient. This is the rationale for making available independent specifications of semantics which may be equivalent. A similar rationale applies for presenting a variety of different specifications for the same semantics.

To illustrate these definitions, note that the state sequence semantics determines the standard relational semantics, since $Tr_a = Tr_b$ clearly implies $R_a = R_b$. It should also be clear that they are not equivalent: *NOP* and *NOP;NOP* are the two most trivial programs with distinct state sequence meanings but the same standard relational meaning.

We will show that the standard partial correctness semantics considered in Section 4, two out of three of the predicate transformer semantics in Section 5, and the standard relational semantics are equivalent, but *wp* (Dijkstra's weakest pre-condition predicate transformer) and the standard relational semantics are incomparable, i.e., neither determines the other.

3. Relational Semantics

In this section we will consider several alternative specifications of the standard relational semantics, **R**, which associates with each program a the relation R_a defined in Section 2. More generally, an arbitrary *relational semantics* **M** is any mapping which assigns to each program a some relation $M_a \subset S \times S$.

3.1 An Inductive Definition

A simple definition of the relation R_a to be associated with any program a can be given by induction on the syntax of programs, using only familiar mathematical operations on relations. In order to do this it is convenient to define R_p for any predicate expression p to be $\{(s, s) \mid P(s)\}$. For $R_1, R_2 \subset S \times S$, let R_1^* be the reflexive transitive closure of R_1, and $R_1 \circ R_2$ the composition of R_1 and R_2. We assume that relations R_A for each primitive statement A are given. Then the relations associated with programs can be defined as follows:

R1. $R_{NOP} = \{(s,s) \mid s \in S\}$ = the identity on S,

R2. $R_{a;b} = R_a \circ R_b$,

R3. $R_{\textit{while}\ p\ \textit{do}\ a} = (R_p \circ R_a)^* \circ R_{\neg p}$.

This specification is trivially derived from TR1-3 given in Section 2; obviously R1-3 specifies directly the same standard relational semantics that TR1-3 specified indirectly.

3.2 Some Axioms for the Standard Relational Semantics

Hoare and Lauer choose to specify the standard relational semantics by giving a system of axioms for statements of the form "started in state s, program a terminates in state s'." We shall refer to such assertions as "transition assertions", and follow Hoare and Lauer in using the notation $s(a)s'$ to denote such a statement. Thus,

Definition 1: $s(a)s'$ is *true for* $\mathbf{M}$ *iff* $(s, s') \in M_a$, where $\mathbf{M}$ is an arbitrary relational semantics.

Their axioms[5] are as follows:

HL1. $s(A)s' \leftrightarrow (s, s') \in R_A$,

HL2. $s(a;b)s' \leftrightarrow \exists t[s(a)t \wedge t(b)s']$,

HL3. $s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s' \rightarrow \neg P(s')$,

HL4. $\forall s_1, s_2[(Q(s_1) \wedge P(s_1) \wedge s_1(a)s_2) \rightarrow Q(s_2)] \rightarrow$
$[(Q(s) \wedge s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s') \rightarrow Q(s')]$,

HL5. $s(NOP)s' \leftrightarrow s = s'$.

They go on to prove that the standard relational semantics $\mathbf{R}$ is a model of HL1-5, that is, every instance of HL1-5 is true for $\mathbf{R}$, so that any conclusion which logically follows from these axioms will be true of the standard semantics.

Of course this meets only half the requirements for specifying the semantics, since one must also show that any transition assertion which is true of the standard semantics follows logically from the axioms. Unfortunately HL1-5 do not imply all the true assertions, contrary to the "intuitive confidence in the completeness of the theory" expressed by Hoare and Lauer [p. 144], as we now illustrate.

We can understand the significance of HL1-5 as follows. If $\mathbf{M}$ is a model of H1, we can conclude that $M_A = R_A$ for each atomic statement A. Similarly, from HL5 we conclude that M_{NOP} = the identity on $S = R_{NOP}$, and from HL2 that $M_{a;b} = M_a \circ M_b$. It follows that $M_a = R_a$ for every *while*-free program a whenever $\mathbf{M}$ is a model of HL1, 2, 5.

Now consider the particular "divergent loop" relational semantics $\mathbf{L}$ defined as follows:

$L_a = R_a$ if a is *while*-free,

$L_a = \phi$ otherwise.

Then $\mathbf{L}$ is obviously a model of HL1, 2, 5. But $s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s'$ is always false for $\mathbf{L}$, so HL3-4 are true, vacuously, for $\mathbf{L}$. Hence $\mathbf{L}$ is also a model of HL1-5.

The divergent loop semantics corresponds to an implementation in which the interpreter simply loops unconditionally whenever it starts to execute a *while* statement. Since $\mathbf{L}$ is a model, statements which logically follow from HL1-5 must always be true of this implementation. In particular, no transition assertion involving a program containing a *while*-loop follows from HL1-5, and so it seems hard to imagine circumstances in which HL1-5 would serve as an adequate characterization of the standard semantics. (However, in Section 4.4 we try to set matters right by indicating a sense in which HL1-5 do in fact specify $\mathbf{R}$.)

3.3 A Complete Set of Axioms for the Standard Relational Semantics

There is no inherent obstacle to presenting axioms in the spirit of HL1-5 which correctly and completely specify the intended semantics. Indeed, adding two more axioms will suffice:

HL6. $\neg P(s) \rightarrow s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s$,

HL7. $[P(s) \wedge s(a)s' \wedge s'(\textit{while}\ \mathrm{p}\ \textit{do}\ a)t] \rightarrow s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)t$.

It is easy to verify that the standard semantics is a model of HL1-7. In Appendix A we prove:

Theorem 1: The standard relational semantics is the only model of HL1-7.

We remark that HL1-7 can be shown to be independent, i.e., Theorem 1 is not true when any one of HL1-7 is omitted.

3.4 A Deductive System for the Standard Relational Semantics

Another, perhaps more straightforward, way to specify the standard relational semantics is to give a system of axioms and inference rules for deducing transition statements. One such system is:

Axioms:

T1. $s(A)s'$, for all $s, s' \in S$ such that $(s, s') \in R_A$,

T2. $s(NOP)s$,

T3. $s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s$, for all $s \in S$ such that $\neg P(s)$.

Inference Rules:

T4. $s(a)t, t(b)s' \vdash s(a;b)s'$,

T5. $s(a)t, t(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s' \vdash s(\textit{while}\ \mathrm{p}\ \textit{do}\ a)s'$, for all $s \in S$ such that $P(s)$.

Let Th(TI-5) be the set of transition statements provable from TI-3 using T4-5.

Lemma 1: $R_a = \{(s, s') \mid s(a)s' \in Th(TI\text{-}5)\}$.

The proof, which we omit, is a routine induction on the structure of a and the number of executions of the body of a *while*-loop. (*Cf.* Appendix E, however, for a similar proof for a more general deductive system.)

Thus, the deductive system TI-5 specifies the same relational semantics as RI-3, and either can serve as the definitive specification. (We caution the reader not to confuse this deductive *specification* of a relational semantics, with the deductive "theory" of Hoare and Lauer which we treat in Section 4.2 as a specification of a partial correctness semantics.)

The specification of R_a in terms of Th(TI-5) given in Lemma 1 can be rephrased in terms of familiar properties of deductive systems. Namely, TI-5 is *sound* for $\mathbf{R}$, which means that every termination assertion in Th(TI-5) is true for $\mathbf{R}$, and TI-5 is *complete* for $\mathbf{R}$, which means that every termination assertion true for $\mathbf{R}$ is in Th(TI-5). Thus we can restate Lemma 1 as

Theorem 2: The set of transition assertions derivable in the system TI-5 is equal to the set of transition assertions true for the standard relational semantics.

4. Partial-Correctness Specifications and Semantics

Assertions of the form "*if* P holds before executing a, then if and when a halts, Q will hold" occur frequently when the behavior of programs is being described. Such assertions are called *partial correctness assertions* (pca's) and are abbreviated $P\{a\}Q$.

We shall define a *partial correctness semantics* for our programming language to be any mapping which assigns to each program a some binary relation on predicates. Any relational semantics $\mathbf{M}$ naturally determines a corresponding partial correctness semantics $\underline{\mathbf{M}}$ which assigns to program a the relation $\underline{\mathbf{M}}_a$ consisting of those pairs (P, Q) such that $P\{a\}Q$ is true of $\mathbf{M}$.[6]

We shall observe that every relational semantics is equivalent to its associated partial correctness semantics. We give a complete deductive system for pca's and an axiom system for pca's. The pca's will serve as specifications of partial correctness semantics as well as specifications of relational semantics. The significance of specifications which have many relational models is considered, and we analyze several such specifications.

4.1 The Standard Partial Correctness Semantics

Definition 2: A partial correctness assertion consists of a program a and a pair (P, Q) of predicates on states, and is written "$P\{a\}Q$." The pair (P, Q) *holds* for a binary relation, R, on states

iff $\forall s, s'[(P(s) \wedge (s, s') \in R) \rightarrow Q(s')]$. $P\{a\}Q$ is *true for a relational semantics* $\mathbf{M}$ *iff* (P, Q) holds for M_a.

The partial correctness semantics $\underline{\mathscr{R}}$ in which $\mathscr{R}_a = \{(P, Q) \mid P\{a\}Q \text{ is true for } \mathbf{R}\}$ is called *the standard partial correctness semantics.*

An arbitrary relation $\mathscr{M}$ on predicates also determines a relation on states in a natural way. The relation is the maximum relation, M, such that all the pairs in $\mathscr{M}$ hold for M. (That there always is such a maximum relation is shown in Appendix B, Lemma B1.) The rationale for taking this relation on states to be the one determined by a partial correctness semantics is nicely expressed by Schwarz [1974, p. 28]:

> "Asserting a partial correctness statement is essentially asserting that certain environments are not the results of executing some command starting in certain other environments. . This is a negative requirement, it does not force any environment to be the result of any execution. Since this is the inherent nature of the formalism it indicates that the proper kind of definition of the semantics determined by a system should have the form: 'largest possible semantics.' "

Definition 3: Let $\mathscr{M}$ be a binary relation on predicates. Then

$$max(\mathscr{M}) = \{(s, t) \mid P(s) \rightarrow Q(t) \text{ for all } (P, Q) \in \mathscr{M}\}.$$

We prove in Appendix B that $max(\mathscr{M})$ is indeed the maximum relation on states for which all the pairs of predicates in $\mathscr{M}$ hold. Moreover, we prove

Lemma 2: Let $\mathbf{M}$ be a relational semantics. Then $M_a = max\{(P, Q) \mid P\{a\}Q \text{ is true for } \mathbf{M}\}$.

An immediate consequence of Lemma 2 is that $R_a = max(\mathscr{R}_a)$, which implies that the standard partial correctness semantics determines the standard relational semantics.[7] The converse determination follows by definition of $\mathscr{R}_a$, namely, $\mathscr{R}_a = \{(P, Q) \mid (P, Q) \text{ holds for } R_a\}$. Thus we have

Theorem 3: The standard relational and standard partial correctness semantics are equivalent.

This theorem and the underlying Lemma 2 provide formal justification for the thesis that the initial-state final-state behavior of programs can be specified by the set of pca's true of the programs.[8]

4.2 Deducing Partial Correctness

The standard partial correctness semantics can, like the standard relational semantics, be specified by a simple system of axioms and inference rules. The notion of the *weakest antecedent*,

$[R]Q$, of a predicate Q under a relation R is used in the axioms for primitive instructions. Informally, $[R]Q$ is the predicate on states which is true of a state s provided that, if and when a program with initial-state, final-state relation R halts after being started in s, the predicate Q will *necessarily* hold.[9]

It is worth emphasizing that we will keep to the usual mathematical conventions in the vacuous case, namely, if the program does not halt started in state s, then $[R]Q$ is true of s for any predicate Q.

Definition 4: Let R be a binary relation on states. For any predicate Q on states, the *weakest antecedent* of Q under R is a predicate, $[R]Q$, on states defined by

$$([R]Q)(s) \textit{ iff } (\forall s')[(s,s') \in R \rightarrow Q(s')].$$

It follows immediately from Definitions 2 and 4 that $([M_a]Q)\{a\}Q$ is true for any relational semantics $\mathbf{M}$ which is why $[M_a]Q$ is called an "antecedent" of Q.

We shall use the notation "$\models(P \rightarrow Q)$" to mean that predicate P implies predicate Q, that is, $\forall s(P(s) \rightarrow Q(s))$. The following lemma explains why $[M_a]Q$ is called "weakest."

Lemma 3: $P\{a\}Q$ is true for $\mathbf{M}$ *iff* $\models(P \rightarrow [M_a]Q)$.

The proof follows directly from the definitions and is omitted (*cf.* [Pratt, 1976; Harel, Meyer, Pratt, 1975; Schwarz, 1974]).

The following system is usually referred to as the Floyd-Hoare system for partial correctness.

Axioms:

FH1. $P\{NOP\}P$,

FH2. $([R_A]Q)\{A\}Q$,

Inference Rules:

FH3. $P\{a\}P'$, $P'\{b\}Q \vdash P\{a;b\}Q$,

FH4. $(P \wedge Q)\{a\}Q \vdash Q\{\textit{while } p \textit{ do } a\}(Q \wedge \neg P)$,

FH5. $P\{a\}Q \vdash (P \wedge P')\{a\}(Q \vee Q')$.

Let Th(FH1-5) be the set of pca's derivable from FH1-2 using FH3-5. We prove in Appendix C, that FH1-5 specifies the standard partial correctness semantics. Formally, we can state

Lemma 4: $\mathscr{R}_a = \{(P, Q) \mid P\{a\}Q \in Th(FHI\text{-}5)\}$.

We have formulated Lemma 4 to emphasize our view of the system FHI-5 as a specification of a mathematical object, namely, a partial correctness semantics which turns out to be the standard one. As we did earlier for termination assertions we can also rephrase Lemma 4 from the more familiar viewpoint that truth of pca's is to be reckoned relative to the standard relational semantics. Then Lemma 4 means that FHI-5 is a sound and complete deductive system for pca's, that is,

Theorem 4: The set of partial correctness assertions derivable from FHI-5 is equal to the set of partial correctness assertions true for the standard relational semantics.

The system FHI-4 consisting of the first four of the Floyd-Hoare rules corresponds to the Deductive Theory[10] DI-3 of Hoare and Lauer [p. 146]. The system FHI-4 is *not* complete, but we will see in Section 4.5 that there is a sense in which the incomplete system FHI-4 specifies the standard relational semantics.[11]

4.3 Axioms for Partial Correctness Semantics

Although a deductive system resembling FHI-5 is the more usual specification of the standard partial correctness semantics, we can also write an axiom system to specify it. The axioms are suggested straightforwardly by the deductive system.

PC1. $P\{NOP\}Q \leftrightarrow \models(P \rightarrow Q)$,

PC2. $P\{A\}Q \leftrightarrow \models(P \rightarrow [R_A]Q)$,

PC3. $P\{a;b\}Q \leftrightarrow \exists P'(P\{a\}P' \wedge P'\{b\}Q)$,

PC4. $Q\{\textit{while } p \textit{ do } a\}Q' \leftrightarrow \exists Q''[\,[(P \wedge Q'')\{a\}Q''] \wedge \models(Q \rightarrow Q'') \wedge$

$\models((Q'' \wedge \neg P) \rightarrow Q')\,]$.

To say how these axioms specify the partial correctness semantics we recall the technical meaning of the word *model* and distinguish two special kinds of models.

A mathematical object is said to be a *model* for a set of assertions if all the assertions are true for the object. We have already used this notion in Section 3 where the objects were relational semantics and the assertions were transition assertions. By Definition 2 we know what it means for a pca to be true of a relational semantics, and hence we know when a relational semantics is a *relational model* for a set of assertions (such as PCI-4) involving pca's. We can also regard a pca as making an assertion about partial correctness semantics.

Definition 5: $P\{a\}Q$ is *true for* a partial correctness semantics $\mathscr{M}$ *iff* $(P, Q) \in \mathscr{M}_a$. A partial

correctness semantics is a *partial correctness model* for an axiom system (such as PCI-4) *iff* it is a model for the set of all instances of those axioms.

Note that Definitions 2 and 5 are compatible in that if $\mathbf{M}$ is any relational semantics and $\underline{\mathcal{M}}$ is the corresponding partial correctness, then P{*a*}Q is true for $\mathbf{M}$ *iff* it is true for $\underline{\mathcal{M}}$. In particular P{*a*}Q has the same truth value in both the standard relational and partial correctness semantics.

Theorem 5(deBakker[12]): The standard partial correctness semantics is the only partial correctness model of PCI-4.

The proof is in Appendix D.

Again, we have formulated this theorem to emphasize our view of PCI-4 as uniquely specifying a particular partial correctness semantics.

4.4 Relational Models for Partial Correctness Specifications

We have just considered FHI-5 and PCI-4 as direct specifications of partial correctness semantics. We now take the more usual view and consider FHI-5 and PCI-4 as specifications of relational semantics according to their relational models. Thus we can rephrase Theorems 2, 4 and 5 in part by saying that $\mathbf{R}$ is a relational model of TI-5, FHI-5 and PCI-4.[13]

Notice that despite Theorems 2 and 4, we cannot say that $\mathbf{R}$ is the only model of TI-5 or FHI-5. For example, the "empty" semantics which assigns the empty relation to every program is a model of FHI-5, and the semantics which assigns the "total" relation S×S to every program is a model of TI-5.

A set of pca's will generally fail to have a unique relational model because, as suggested by the quotation in Section 4.1, pca's are "anti-monotone" in the following sense. If $\mathbf{M}$ and $\mathbf{N}$ are relational semantics then we shall say that $\mathbf{N}$ is larger than $\mathbf{M}$ *iff* $N_a \supset M_a$ for all programs *a*. Then by Definition 2 we see that if P{*a*}Q is true for $\mathbf{N}$, and $\mathbf{N}$ is larger than $\mathbf{M}$, then P{*a*}Q is also true for $\mathbf{M}$. Thus, since $\mathbf{R}$ is a model of FHI-5, so is any relational semantics smaller than $\mathbf{R}$.[14]

On the other hand, Theorem 4 and Lemma 2 together imply that $\mathbf{R}$ is larger than any model of FHI-5, so we can conclude

Theorem 6: The standard relational semantics is the largest relational model of FHI-5.

Similarly, transition assertions are "monotone" in the sense that if $s(a)s'$ is true for $\mathbf{M}$, and $\mathbf{N}$ is larger than $\mathbf{M}$, then $s(a)s'$ is true for $\mathbf{N}$. We conclude from Theorem 2 that

Theorem 7: The standard relational semantics is the smallest model of TI-5.

Finally, we can deduce from Theorem 5 that

Theorem 8: The standard relational semantics is the only relational model of PCI-4.

Thus, Theorems 6, 7, and 8 reveal precisely the different ways in which $\mathbf{R}$ is determined uniquely by the specifications FHI-5, TI-5, PCI-4.

We should point out that Theorem 8 is technically a slightly weaker result than Theorem 5. Theorem 8 in effect asserts that among the partial correctness semantics which are determined by relational semantics, only the partial correctness semantics $\mathcal{R}$ determined by $\mathbf{R}$ is a model of PCI-4. On the other hand, Theorem 5 asserts that among *all* partial correctness semantics, not just those determined by relations, $\mathcal{R}$ is the unique model.

4.5 Implications Between Semantical Specifications with Several Models

The need to deal with specifications having several models of a given kind was allowed for by Hoare and Lauer in their formulation of what they call "consistency" between semantical specifications. They say that one specification is consistent with another *iff* every model of the latter is a model of the former.

Notice that this definition is asymmetrical, and so conflicts with ordinary usage of the word "consistency." For this reason, we shall refer to "implication" between specifications, that is, specification $\mathcal{S}$ implies specification $\mathcal{T}$ *iff* every model of $\mathcal{S}$ is a model of $\mathcal{T}$.

Semantical specifications with more than one model can be useful. We have just seen that while FHI-5 and TI-5 technically speaking have many relational models, nevertheless they uniquely specify $\mathbf{R}$ in a natural way as the largest relational model and smallest relational model, respectively. Even more generally there may be situations in which any of several models would suffice for some application, and we wish only to specify this set of appropriate models -- not necessarily distinguishing a canonical model in the set by some criterion such as maximality. For example, in the specification of practical programming languages it is typical to leave undefined the meaning of certain syntactically well-formed programs. In such cases there will be many acceptable semantics differing only on the meanings, e.g., error messages, they assign to "meaningless" programs.

In addition to the axioms HLI-5 considered above, Hoare and Lauer offer the first four rules FHI-4 of the Floyd-Hoare system (*cf.* footnote 10) as a specification with multiple relational models, and they seem to suggest that these multiple models represent possible acceptable semantics. However, when we look more closely at the Hoare-Lauer and Floyd-Hoare axioms we shall see that an example of a specification which could be met by many acceptable semantics does not arise here; there is only one intended model of these particular specifications, although it takes some effort to discover the sense in which these specifications determine that model.

In particular, Hoare and Lauer [Theorem 4] observe that HLI-5 implies the first four Floyd-Hoare rules FHI-4.[15] For some reason they do not consider the converse question of whether FHI-4 implies HLI-5. In fact, it does not; not even the full Floyd-Hoare system FHI-5 implies HLI-5. This is because any $\mathbf{M}$ smaller than $\mathbf{R}$ is a relational model of FHI-5, so that, for example, the empty semantics is a model of FHI-5 but not of HLI-5.

However, Hoare and Lauer's proof that HLI-5 implies FHI-4 actually establishes a slightly stronger result which we can use to reveal the connections between HLI-5, FHI-4, and $\mathbf{R}$.

An inference rule such as any of FH3-5 or T4-5 will be called *sound* for a relational semantics $\mathbf{M}$, if, whenever the conditions (such as those for T5) for applicability of the rule are satisfied and the antecedent(s) of the rule is true for $\mathbf{M}$, so is the consequent. In other words, an inference rule is sound if application of it preserves truth.

Lemma 5: If $\mathbf{M}$ is a model of FHI-2 and the inference rules FH3-4 are sound for $\mathbf{M}$, then $\mathbf{M}$ is a model of FHI-5.

Proof: It is easy to see that FH5 is sound for all $\mathbf{M}$.∎

Theorem 9: $\mathbf{R}$ is the largest relational model of FHI-2 for which the inference rules FH3-4 are sound.

Proof: We let the reader convince himself that FH3-4 are sound for the standard relational semantics $\mathbf{R}$ (*cf.* [Hoare and Lauer, Theorem 4]). Thus, $\mathbf{R}$ is "a" model; that it is "the largest" model is immediate from Theorem 6 and Lemma 5.∎

Lemma 6(Hoare and Lauer): Let $\mathbf{M}$ be a model of HLI-5. Then $\mathbf{M}$ is a model of FHI-2 and the inference rules FH3-4 are sound for $\mathbf{M}$.

We shall not repeat the proof (*cf.* [Hoare and Lauer, page 147]).

Theorem 10: $\mathbf{R}$ is the largest model of HLI-5.

Proof: Immediate from Theorem 9 and Lemma 6.∎

The preceding theorems thus reveal the sense in which HLI-5 and the first four Floyd-Hoare rules FHI-4 serve as semantical specifications equivalent to the others we have considered -- a rather obscure technical sense which was left implicit by Hoare and Lauer.[16]

Our point here is that while we agree with Hoare and Lauer that relationships like implications between specifications with multiple models are important ideas, it is even more important to have a clear understanding of the family of models which are to be regarded as meeting the specifications. This is illustrated by the fact that the semantics $\mathbf{L}$ of Section 3.2 is a relational model both of HLI-5 and FHI-5, yet we certainly do not mean to accept an implementation of our language in which all *while*-loops diverge.

5. Predicate Transformers

There is yet another kind of semantics found in the literature, namely predicate transformer semantics. Instead of assigning a set of assertions to a program as its meaning, one can assign a function on predicates, called a predicate transformer, to that program. An example of a predicate transformer is [R] for any binary relation R on states. This transformer maps each predicate Q into its weakest antecedent [R]Q. Another useful transformer is <R>, defined as transforming any predicate Q into the predicate $\neg[R]\neg Q$.

The transformer which has received much attention recently is Dijkstra's weakest pre-condition wp_a. The predicate $wp_a(Q)$ is described by Dijkstra [1976, p. 16] as

> "the condition which characterizes *all* initial states such that activation will certainly result in a properly terminating happening, leaving the system in a final state satisfying [the condition Q] ..."

We shall observe that for *while*-programs, if the primitive instructions A are well-behaved, e.g., if the relations R_A are in fact functions, then $wp_a = \langle R_a \rangle$. This will enable us to conclude that in such cases wp yields a predicate transformer semantics which is equivalent to the standard semantics. However in the general case when nondeterministic primitive instructions occur, wp semantics is incomparable to, i.e., it neither determines nor is determined by, the standard semantics.

As with other kinds of semantics, predicate transformers can be specified in several ways. We exhibit inductive definitions and deductive systems specifying wp and <R>.

5.1 The Weakest Antecedent and Possible Termination Transformers

The *standard weakest antecedent transformer semantics* associates to each program a the meaning $[R_a]$. Thus by definition it is determined by the standard relational semantics. Conversely, observing that (P, Q) holds for R *iff* $\models P \rightarrow [R]Q$ (*cf.* Lemma 3), it follows that the standard partial correctness semantics is determined by the standard weakest antecedent transformer. The same observation reveals that we may define the weakest antecedent directly in terms of pca's as follows:

$$[M_a]Q = \vee\{P \mid P\{a\}Q \text{ is true for } \mathbf{M}\}.^{17}$$

Thus, the predicate transformer semantics based on weakest antecedent carries the same information as relational and partial correctness semantics.

The predicate $\langle R\rangle Q$, defined to be $\neg[R]\neg Q$, can be described informally as being true of a state s providing that: it is always *possible* starting in state s, to execute a (generally nondeterministic) program with initial-state, final-state relation R and halt in a state in which Q is

true. We refer to $\langle R \rangle$ as the *possible termination transformer* corresponding to R and define the *standard possible termination transformer* as associating $\langle R_a \rangle$ with a. Again, since $[R]Q = \neg\langle R \rangle \neg Q$, we see that possible termination semantics carries the same information as weakest antecedent semantics.

To summarize, we can state

Theorem 11: The standard weakest antecedent transformer, possible termination transformer, partial correctness, and relational semantics are all equivalent.

5.2 Inductive Definitions of the Weakest Pre-condition and Possible Termination Transformers

For the class of *while* programs we consider, Dijkstra [1976][18] clarifies the informal description of weakest pre-conditions quoted above by giving an inductive definition:

WP1. $wp_{NOP}(Q) = Q$.

WP2. $wp_{a;b} = wp_a(wp_b(Q))$.

WP3. $wp_{while\ p\ do\ a}(Q) = \bigvee_k H_k$

where $H_0 = \neg P \wedge Q$, and
$H_{k+1} = (P \wedge wp_a(H_k)) \vee H_0$.

If we assume that wp_A is given for all primitive instructions A, then WP1-3 uniquely define the weakest pre-condition predicate transformer. In particular, when, as is typically the case, the primitive instructions are functional, we expect that $wp_A = \langle R_A \rangle$.[19] The intuition behind this latter equation is that it is appropriate to treat an instruction as primitive only if it is sure to terminate whenever there is a legal termination state. It follows that if an instruction can *possibly* terminate in a state satisfying Q, i.e., if $\langle R_A \rangle Q$ holds, then because it is a primitive instruction, it will *certainly* terminate in some state, and because it is functional this state of termination is unique and satisfies Q, i.e., $wp_A(Q)$ holds. In this situation, it turns out that weakest pre-condition and possible termination semantics coincide for *all* programs of the simple kind we have been considering.[20]

Lemma 7: If $wp_A = \langle R_A \rangle$ for all primitive instructions A, then $wp_a = \langle R_a \rangle$ for all programs a.

The proof, which we omit, follows directly by induction on the structure of programs from WP1-3 and the definition of $\langle R_a \rangle$.

As a consequence of Lemma 7, we can replace "wp" by "$\langle R \rangle$" throughout WP1-3 thereby obtaining an inductive definition of $\langle R_a \rangle$.

5.3 Deductive Specification of Possible Termination

Assertions of the form "if P holds before executing a, then it is possible for a to halt with Q holding true" are called *termination assertions* and are abbreviated $P\langle a\rangle Q$. Termination assertions correspond to possible termination transformers in the same way that pca's correspond to weakest antecedents.

Definition 6: $P\langle a\rangle Q$ is *true for the relational semantics* $\mathbf{M}$ *iff* $\models(P \rightarrow \langle M_a\rangle Q)$.

Thus, we have immediately that

$$\langle M_a\rangle Q = \vee\{P \mid P\langle a\rangle Q \text{ is true for } \mathbf{M}\},$$

and we can regard a deductive system for termination assertions as providing a specification of the possible termination transformer.

We remark that termination assertions are a natural generalization of the transition assertions of Section 3.2; the transition assertion $s\langle a\rangle t$ is equivalent to the termination assertion $E_s\langle a\rangle E_t$, where E_s is the "equals s" predicate true only in state s. The following deductive system correspondingly generalizes TI-5.

Axioms:

TA1. $P\langle NOP\rangle P$,

TA2. $(\langle R_A\rangle Q)\langle A\rangle Q$

TA3. $(\neg P \wedge Q)\langle$*while* p *do* $a\rangle Q$;

Inference Rules:

TA4. $P\langle a\rangle Q'$, $Q'\langle b\rangle Q \vdash P\langle a;b\rangle Q$;

TA5. $Q\langle a\rangle Q''$, $Q''\langle$*while* p *do* $a\rangle Q' \vdash (P \wedge Q)\langle$*while* p *do* $a\rangle(Q' \wedge \neg P)$;

TA6. $P\langle a\rangle Q \vdash (P' \wedge P)\langle a\rangle(Q \vee Q')$;

TA7. $P_1\langle a\rangle Q$, $P_2\langle a\rangle Q \vdash (P_1 \vee P_2)\langle a\rangle Q$.

It is easy to see that TA1-7 is sound for the standard relational semantics $\mathbf{R}$. Although not all termination assertions true for $\mathbf{R}$ are derivable, viz., TA1-7 is not quite complete, those assertions whose antecedents are *finite* are derivable.

Theorem 12: Let P be a predicate true in only finitely many states. Then $P\langle a\rangle Q$ is derivable from TAI-7 *iff* $P\langle a\rangle Q$ is true for the standard relational semantics.

It follows immediately that TAI-7 suffices to determine the standard possible termination transformers.

Corollary 13: $\langle R_a\rangle Q = \vee\{P \mid P\langle a\rangle Q \in Th(TAI\text{-}7)\}$.

It is interesting to note that a complete system can be obtained by extending TA7 to an infinitary rule

TA8. $\{P_i\langle a\rangle Q \mid i \in I\} \vdash (\vee\{P_i \mid i \in I\})\langle a\rangle Q$

where I is any index set.

Theorem 14: The termination assertions derivable from TAI-8 are precisely those true for the standard relational semantics.

The proofs of Theorems 12 - 14 are in Appendix E.[21]

5.4 Weakest Pre-conditions of Nondeterministic Programs

In contrast to all the kinds of semantics considered so far, weakest pre-conditions for programs with nonfunctional primitive instructions reflect an understanding of the meaning of programs which is neither determined by nor determines the meaning given by the standard semantics.

To illustrate these differences, let A_1 be *NOP*, let A_2 be a primitive instruction which resets every state to a given state s_0, and let A_3 be a primitive instruction which makes a nondeterministic choice between behaving like A_1 or A_2. Thus,

R_{A_1} = I, the identity on states,
$R_{A_2} = S \times \{s_0\}$, the constant function mapping any state s to s_0, and
$R_{A_3} = R_{A_1} \cup R_{A_2}$.

Since A_1 and A_2 are primitive and functional, we define

$wp_{A_1}(Q) = \langle R_{A_1}\rangle Q = Q$, and
$wp_{A_2}(Q) = \langle R_{A_2}\rangle Q$ = the constant predicate with value $Q(s_0)$.

In order to define the weakest pre-condition transformer for a nonfunctional primitive instruction such as A_3, we rely on Dijkstra's English description given above, combined with the intuitive reasoning indicated before Lemma 7. Namely, the pre-condition which "will certainly

result" in post-condition Q after execution of a primitive instruction A is $[R_A]Q$, providing that execution terminates; for a *primitive* instruction we expect execution to terminate whenever there is a final state related to the initial state, that is, whenever $\langle R_A \rangle true$ holds. Thus we define

$$\begin{aligned} wp_{A_3}(Q) &= [R_{A_3}]Q \wedge \langle R_{A_3} \rangle true \\ &= [R_{A_3}]Q \wedge true = [R_{A_3}]Q \\ &= Q \wedge Q(s_0).^{22} \end{aligned}$$

Let a_i be the program *while* p *do* A_i, for i = 1, 2, 3 and $P = \neg E_{s_0}$. Then it follows from R1-3 that

$$R_{a_1} = R_{\neg p} = \{(s_0, s_0)\}, \text{ and}$$

$$R_{a_2} = R_{a_3} = R_{A_2} = S \times \{s_0\}.$$

On the other hand, it follows from WP3 that

$$wp_{a_1}(Q) = wp_{a_3}(Q) = E_{s_0} \wedge Q(s_0), \text{ and}$$

$$wp_{a_2}(Q) = wp_{A_2}(Q) = Q(s_0).$$

Now we see that the initial-state, final-state relation of a program cannot in general determine its weakest pre-condition transformer, because a_2 and a_3 are assigned the same relation by the standard relational semantics, but define distinct wp transformers. Conversely, the wp transformer of a program cannot in general determine its initial-state, final-state relation, because a_1 and a_3 have the same wp transformer, but are assigned different relations.

The reason for these discrepancies is, roughly speaking, that the intended interpretation of how a program "can *certainly* [emphasis added] result in a properly terminating happening" reflected in WP1-3 requires, in addition to halting states being possibly accessible and all such states satisfying proper post-conditions, that there be no possibility of "looping" or "failing" branches among the various courses of a nondeterministic computation (*cf.* [Harel and Pratt, 1978; Harel, 1978; Hoare, 1978]). Programs a_2 and a_3 differ in that a_3 allows a possibility of infinite looping, so that wp_{a_2} differs from wp_{a_3} even though $R_{a_2} = R_{a_3}$. Similarly, even though a_3 *can* halt on every state and a_1 halts only on s_0, from the point of view of *certainty* of proper termination, a_3 is no better than a_1 because a_3 allows looping in every state other than s_0; this is reflected in the fact that $wp_{a_1} = wp_{a_3}$ even though $R_{a_1} \neq R_{a_3}$.

It is possible to extend the notion of relational semantics so that looping or failing is explicitly indicated by the presence of a special state, $\perp$, with certain algebraic properties with respect to the other states. With some care, an extended relational semantics $\mathbf{R}^{\perp}$ can be defined inductively with the result that the extended initial-state, final-state relation of a program does indeed determine its wp (*cf.* [deBakker, 1978]). However, the converse difficulty, that wp_a does not determine either R_a or $R_a^{\perp}$ remains.

Satisfactory deductive or axiomatic characterizations of wp for nondeterministic programs have proven difficult to devise. For this reason among others, we are inclined to agree with the arguments in [Harel, 1978; Harel and Pratt, 1978] that the semantical ideas implicit in wp are better treated by considering weakest antecedents, possible termination, looping, and failing, as four separate notions.

6. Conclusion

We have mainly looked at three kinds of semantics -- relational, partial correctness and predicate transformer -- and several ways of specifying a semantics -- inductive definitions, axiom systems, deduction systems. Each semantics was specified with roughly equal economy and complete precision in several of these ways. There was no particular technical problem in defining rigorously how specifications determined semantics, although there were three or four different mathematical mechanisms used to connect the specifications with the intended semantics.

The standard relational and partial correctness semantics are equivalent. This means that the set of all partial correctness assertions true for our trivial programming language gives exactly the same information as the relational semantics. (This is true despite the fact that in a certain narrow technical sense partial correctness assertions cannot be used to express termination of programs.) Either kind of semantics can be specified directly using an axiom system or a deductive system; either semantics determines the other, independent of means of specification. For deterministic programs, similar observations were made about predicate transformer semantics. However, for nondeterministic programs the predicate transformer wp_a may not determine or be determined by the initial-state, final-state relation of program a. Satisfactory axiomatic alternatives to the inductive specification of wp_a have not been found.

It may be worth remarking that the entire preceding development extends easily to the somewhat richer programming language considered by Lauer [1971] including conditional and nondeterministic choice statements, blocks with local variables, and nonrecursive procedure declarations and calls.

Syntax played a limited role in this paper. Only programs were syntactic objects; predicates were treated as mathematical, set-theoretic objects. The next refinement of the study begun here involves restricting predicates to those which are definable in some agreed-upon formal notation, e.g., first or second order logics of appropriate structures. When we restrict predicates in this way the situation becomes more complicated -- and more interesting -- and the conclusions we reached above about the equivalence of various kinds of semantics must be modified. Thus, there are cases where the set of all true definable pca's may not determine the proper relational semantics; in other cases a restricted deductive theory may contain only a subset of all true definable pca's and yet determine the right semantics. We postpone to a later paper further discussion of the restriction to definable predicates.

In sum, we have illustrated that, from a purely formal viewpoint, attempting to specify the meaning of a language in several ways can be made to work -- at least for very simple programming languages when we -- unrealistically -- place no restrictions on the language for expressing predicates. However, care had to be taken to indicate how each specification was to be understood before it could be applied by any of the variety of possible users.

Acknowledgements

The authors would like to express their appreciation for the comments and corrections pointed out by Michael J. Fischer, David Harel, C.A.R. Hoare, Peter Lauer, David Park, Rohit Parikh, Michael S. Paterson, Vaughan R. Pratt, Jerald Schwarz, Carl Seaquist, Joel Seiferas, and Peter van Emde Boas.

References

deBakker, J.W. 1975. Flow of Control in the Proof Theory of Structured Programming. *16th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society. Long Beach, Ca. pp 29-33.

deBakker, J.W. 1977. Recursive Programs as Predicate Transformers. *Proc. IFIP Conf. on Formal Description of Programming Concepts* at St. Andrews, New Brunswick. pp 7.1-7.15.

deBakker, J.W. 1978. *Programming Theory.* Unpublished Course Notes. Mathematisch Centrum, Amsterdam.

deBakker, J.W. and L.G.L.T. Meertens. 1975. On the Completeness of the Inductive Assertion Method. *Journal of Computer and Systems Science,* 11, pp 323-357.

Dijkstra, E.W.D. 1975. Guarded Commands, Non-determinacy and Formal Derivation of Programs. *CACM* 18, 8. pp 453-457.

Dijkstra, E.W.D. 1976. *A Discipline of Programming,* Prentice-Hall, Englewood Cliffs, N.J., 217 pp.

Floyd, R.W. 1967. Assigning Meaning to Programs in *Mathematical Aspects of Computer Science. Proceedings of Symposium in Applied Mathematics 19.* (ed. J.T. Schwartz). American Mathematical Society. Providence, R.I. pp 19-32.

Harel, D. 1978. Logics of Programs: Axiomatics and Descriptive Power. Laboratory for Computer Science Technical Report 200. M.I.T., Cambridge, Mass., 152 pp.

Harel, D., A. R. Meyer, and V. R. Pratt. 1977. Computability and Completeness in Logics of Programs. *Proc. of 9th Annual ACM Symposium on Theory of Computing.* IEEE Computer Society. Long Beach, Ca. pp 261-268.

Harel, D. and V. R. Pratt. 1978. Nondeterminism in Logics of Programs. *Conference Record of the Fifth Annual Symposium on Principles of Programming Languages*. ACM, New York, N.Y. pp 203-213.

Hoare, C.A.R. 1978. Some Properties of Predicate Transformers. *JACM* 25, 3. pp 461-480.

Hoare, C.A.R. 1969. An Axiomatic Basis for Computer Programming. *CACM* 12, 10. pp 576-583.

Hoare, C.A.R. and P. Lauer. 1974. Consistent and Complementary Formal Theories of the Semantics of Programming Languages. *Acta Informatica* 3, pp 135-155.

Hoare, C.A.R. and N. Wirth. 1973. An Axiomatic Definition of the Programming Language PASCAL. *Acta Informatica* 2, pp 335-355.

Lauer, P. 1971. Consistent Formal Theories of the Semantics of Programming Languages. Ph.D. Thesis. Faculty of Science, Queen's University of Belfast. Technical Report 25.121, I.B.M. Laboratory Vienna, Austria. 122 pp.

Manna, Z. 1974. *Mathematical Theory of Computation*. McGraw-Hill, New York, 429 pp.

Pratt, V. R. 1976. Semantical Considerations on Floyd-Hoare Logic. *17th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society. Long Beach, Ca. pp 109-121.

Raulefs, Peter. 1977. The Connection Between Axiomatic and Denotational Semantics of Programming Languages. Interner Bericht Nr. 4/77. Institut fur Informatik I. Universitat Karlsruhe. 26 pp.

Schwarz, J.S. 1974. Semantics of Partial Correctness Formalisms. Ph.D. Thesis. Department of Systems and Information Sciences, Syracuse University. Syracuse, N.Y. 126 pp.

specified by a simple system of axioms and inference rules. The notion of the *weakest antecedent*,

Appendix A.

Proof of Theorem 1.

Theorem 1: The standard relational semantics is the only model of HL1-7.

Let T be the set of transition assertions $s(a)s'$ true for a relational model $\mathbf{T}$ of HL1-7. We will prove by induction on a that $(s, s') \in R_a$ *iff* $s(a)s' \in T$. Thus $\mathbf{T} = \mathbf{R}$.

If a is NOP then by HL5, $s(NOP)s' \in T \leftrightarrow s = s' \leftrightarrow (s, s') \in R_{NOP}$. Similarly if a is a primitive statement, A, then by HL1, $s(A)s' \in T \leftrightarrow (s, s') \in R_A$.

If a is $b;c$ then by HL2, induction, and R2,

$s(b;c)s' \in T$

iff $\exists t[s(b)t \in T \wedge t(c)s' \in T]$

iff $\exists t[(s, t) \in R_b \wedge (t, s') \in R_b]$

iff $(s, s') \in R_b \circ R_c$

iff $(s, s') \in R_{b;c}$.

The case of *while* statements follows directly from the following Lemmas A1 and A2.

Lemma A1: If $(s, s') \in R_{while\ \mathrm{p}\ do\ b}$ then $s(while\ \mathrm{p}\ do\ b)s' \in T$.

Proof: We need the following

Definition A1: For states s, s', program b and predicate P, let $dist_{b,\mathrm{P}}(s, s')$ be the least nonnegative integer k, if any, such that there is a sequence $s_0, \ldots, s_k$ of states with the property that

(i) $s_0 = s$,

(ii) $s_k = s'$, and

(iii) $\mathrm{P}(s_i) \wedge (s_i, s_{i+1}) \in R_b$ for all nonnegative integers $i<k$;

if no such k exists the distance $dist_{b,\mathrm{P}}(s,s')$ is said to be infinite.

We take the following two facts as obvious from Definition A1. First, if $dist_{b,\mathrm{P}}(s,s') = n+1$, then P(s) and there is an s_1 such that $(s, s_1) \in R_b$ and $dist_{b,\mathrm{P}}(s_1,s') = n$. Second, $(s,s') \in R_{while\ \mathrm{p}\ do\ b}$ *iff* $dist_{b,\mathrm{P}}(s, s')$ is finite and $\neg\mathrm{P}(s')$.

C, that FHI-5 specifies the standard partial correctness semantics. Formally, we can state

Lemma A1 follows by induction on $dist_{b,p}(s, s')$. If the distance is zero, then $s = s'$ and from the second fact above we conclude that $\neg P(s)$. Then by HL6, $s(\textit{while}\ p\ \textit{do}\ b)s' \in T$.

By the first fact above, if $dist_{b,p}(s, s') = n + 1$ we have P(s) and $(s, s_1) \in R_b$ for some s_1 such that $dist_{b,p}(s_1, s') = n$. From $(s, s_1) \in R_b$, by induction we have $s(b)s_1 \in T$. By induction on n, we have $s_1(\textit{while}\ p\ \textit{do}\ b)s' \in T$. Therefore, by HL7, $s(\textit{while}\ p\ \textit{do}\ b)s' \in T$. ∎(Lemma A1).

Lemma A2: If $s(\textit{while}\ p\ \textit{do}\ b)s' \in T$ then $(s, s') \in R_{\textit{while}\ p\ \textit{do}\ b}$.

Proof: Let $Q(t)$ be the predicate $(s, t) \in (R_p \circ R_b)^*$. We claim that $Q(s_1) \wedge P(s_1) \wedge s_1(b)s_2$ implies $Q(s_2)$. This follows by definition of Q and the fact that

$s_1(b)s_2 \in T$ implies $(s_1, s_2) \in R_b$ by main induction on *a*.

We now have $Q(s)$ by definition, and $s(\textit{while}\ p\ \textit{do}\ b)s' \in T$ by hypothesis. By HL4 and the preceding claim, we can conclude $Q(s')$, and by HL3 and $s(\textit{while}\ p\ \textit{do}\ b)s' \in T$, we have $\neg P(s')$.

Now $Q(s') \wedge \neg P(s')$ implies $(s, s') \in R_{\textit{while}\ p\ \textit{do}\ b}$ by definition of $R_{\textit{while}\ p\ \textit{do}\ b}$. ∎(Lemma A2).

Appendix B.

Proof of Lemma 2

Lemma 2: $M_a = max\ \{(P, Q) \mid P\{a\}Q \text{ is true for } \mathbf{M}\}$.

We first prove the claim that $max(\mathcal{M})$ is the maximum relation for which all pairs of predicates in $\mathcal{M}$ hold.

Definition B1: Let R be a binary relation on states. Then define $\mathcal{M}_R = \{(P, Q) \text{ holds for } R\}$.

Note that all the pairs in $\mathcal{M}$ hold for R *iff* $\mathcal{M} \subset \mathcal{M}_R$. Thus the following lemma establishes the preceding claim.

Lemma B1: $R \subset max(\mathcal{M})$ *iff* $\mathcal{M} \subset \mathcal{M}_R$.

Proof: (only if) Suppose $(P, P') \in \mathcal{M}$. Then by definition of *max*, $P(s) \rightarrow P'(s')$ for all $(s, s') \in max(\mathcal{M})$. Thus, if $R \subset max(\mathcal{M})$, then $P(s) \rightarrow P'(s')$ for all $(s, s') \in R$. That is, by Definition 2, (P, P') holds for R. So $(P, P') \in \mathcal{M}_R$.

(if) Now assume $(s, s') \in R$. By Definition 2, $P(s) \rightarrow P'(s')$ for all $(P, P') \in \mathcal{M}_R$. If $\mathcal{M} \subset \mathcal{M}_R$, then $P(s) \rightarrow P'(s')$ for all $(P, P') \in \mathcal{M}$, and so $(s, s') \in max(\mathcal{M})$ by definition of *max*. ∎(Lemma B1).

Lemma B2: $R = max(\mathcal{M}_R)$.

Proof: $R \subset max(\mathcal{M}_R)$ by Lemma B1. To show equality, suppose $(s_1, s_2) \notin R$. Let E_s be the predicate true only of state s. Then $(E_{s_1}, \neg E_{s_2})$ holds for R, so $(E_{s_1}, \neg E_{s_2}) \in \mathcal{M}_R$ and, by definition of *max*, $(s_1, s_2) \notin max(\mathcal{M}_R)$. ∎(Lemma B2).

Note that if $\mathbf{M}$ is a relational semantics, then $\mathcal{M}_{M_a} = \{(P, Q) \mid (P, Q) \text{ holds for } M_a\} = \{(P, Q) \mid P\{a\}Q \text{ is true for } \mathbf{M}\}$, so Lemma 2 follows immediately from Lemma B2.

Appendix C.

Proof of Theorem 4.

Theorem 4: The set of pca's derivable from FH1-5, is equal to the set of pca's true for the standard relational semantics.

We prove by induction on the structure of programs that P{a}Q true for $\mathbf{R}$ implies P{a}Q derivable.

If P{*NOP*}Q is true for $\mathbf{R}$, then by Definition 2 and R1 we conclude that P implies Q, viz., $\models P \rightarrow Q$, or equivalently, $P \vee Q = Q$. Hence P{*NOP*}Q is derivable by applying FH5 to the FH1 axiom P{*NOP*}P.

If P{*A*}Q is true for $\mathbf{R}$, then by Lemma 3, P implies $[R_A]Q$, so P{*A*}Q is derivable by applying FH5 to the FH2 axiom $([R_A]Q)\{A\}Q$.

If P{*a;b*}Q is true for $\mathbf{R}$, then P{*a*}([R_b]Q) must be true for $\mathbf{R}$, as the reader can verify from Definitions 2, 4, and R2. Also, ([R_b]Q){*b*}Q is true for $\mathbf{R}$ by Definitions 2 and 4. By induction we may conclude that P{*a*}([R_b]Q) and ([R_b]Q){*b*}Q are derivable, and therefore P{*a;b*}Q is derivable by applying FH3.

Finally, suppose $P_1\{$*while* p *do a*$\}P_2$ is true for $\mathbf{R}$. Let $Q = [R_{\textit{while p do } a}]P_2$. Then again it follows directly from the definitions that

(1) P_1 implies Q,

(2) $Q \wedge \neg P$ implies P_2, and

(3) $(Q \wedge P)\{a\}Q$ is true for $\mathbf{R}$.

Then by induction, we conclude from (3) that $(Q \wedge P)\{a\}Q$ is derivable. Applying FH4, we can therefore derive Q{*while* p *do a*}$(Q \wedge \neg P)$. But we can apply FH5 to the latter assertion to derive $(P_1 \wedge Q)\{$*while* p *do a*$\}(P_2 \vee (Q \wedge \neg P))$ which by (1) and (2) is the same as $P_1\{$*while* p *do a*$\}P_2$.

We omit the proof that FH1-5 is sound, i.e., if P{*a*}Q is derivable then P{*a*}Q is true for $\mathbf{R}$.∎

Appendix D.

Proof of Theorem 5.

Theorem 5: $\underline{\mathscr{R}}$ is the only partial correctness model of PCI-4.

The following lemma summarizes some facts about weakest antecedent which are used in the proof.

Lemma D1: Let R, R_1, and R_2 be relations on states.

(a) $\exists P'(\models(P \to [R]P') \wedge \models(P' \to Q))$ *iff* $\models(P \to [R]Q)$.

(b) $[R_1][R_2]Q = [R_1 \circ R_2]Q$.

(c) $\models([R^*]Q \to Q)$.

(d) $\models([R^*]Q \to [R][R^*]Q)$.

Proof of D1: (a). The implication from right to left is trivial since we can choose $P' = Q$. The converse follows from the easily verified fact that $[R](P' \wedge Q) = [R]P' \wedge [R]Q$ (*cf.* [Pratt, 1976]).

(b). Follows from Definition 4. We omit the details (*cf.* [Pratt, 1976]).

(c) and (d). Note that $[R_1 \cup R_2]Q = [R_1]Q \wedge [R_2]Q$. Hence, $[R^*]Q = [I \cup R \circ R^*]Q = [I]Q \wedge [R \circ R^*]Q = Q \wedge [R \circ R^*]Q$. ∎(Lemma D1).

Let $\underline{\mathscr{M}}$ be any partial correctness model of PCI-4. We show that $\mathscr{M}_a = R_a$ by induction on the structure of program a.

$(P, Q) \in \mathscr{M}_{NOP}$ *iff* (by Def. 5) $P\{NOP\}Q$ is true for $\underline{\mathscr{M}}$ *iff* (by PC1) $\models(P \to Q)$ *iff* (by Def. 2 and R1) $(P, Q) \in \mathscr{R}_{NOP}$.

$(P, Q) \in \mathscr{M}_A$ *iff* (by PC2 and Def. 5) $\models(P \to [R_A]Q)$ *iff* (by Defs. 2 and Lemma 3) $(P, Q) \in \mathscr{R}_A$.

Suppose $a = b;c$. Then $(Q, Q') \in \mathscr{M}_a$

iff (by PC3 and Def. 5) $\exists P'((Q, P') \in \mathscr{M}_b \wedge (P', Q') \in \mathscr{M}_c)$

iff (by induction) $\exists P'((Q, P') \in \mathscr{R}_b \wedge (P', Q) \in \mathscr{R}_c)$

iff (by Def. 2 and Lemma 3) $\exists P'((\models Q \to [R_b]P') \wedge \models(P' \to [R_c]Q'))$

iff (by Lemma D1 (a)) $\models(Q \to [R_b][R_c]Q')$

relational model both of HLI-5 and FHI-5, yet we certainly do not mean to accept an implementation of our language in which all *while*-loops diverge.

iff (by Lemma DI (b)) $\models(Q \rightarrow [R_b \circ R_c]Q')$

iff (by R2, Lemma 3, and Def. 2) $(Q, Q') \in \mathcal{R}_a$.

We need the following two lemmas for the case of a = ***while*** p ***do*** b.

Lemma D2: $(Q, Q') \in \mathcal{R}_{while\ p\ do\ b}$ implies

$\exists Q''(\models((Q'' \wedge P) \rightarrow [R_b]Q'') \wedge \models(Q \rightarrow Q'') \wedge \models((Q'' \wedge \neg P) \rightarrow Q'))$.

Proof of D2: $(Q, Q') \in \mathcal{R}_{while\ p\ do\ b}$

iff $\models(Q \rightarrow [R_{while\ p\ do\ b}]Q')$

iff (by R3) $\models(Q \rightarrow [(R_p \circ R_b)^* \circ R_{\neg p}]Q')$

iff (by Lemma DI (b)) $\models(Q \rightarrow [(R_p \circ R_b)^*][R_{\neg p}]Q'$

iff (by Lemma DI (a)) $\exists Q_1(\models(Q \rightarrow [(R_p \circ R_b)^*]Q_1) \wedge \models(Q_1 \rightarrow [R_{\neg p}]Q'))$.

Since $[R_{\neg p}]Q'$ is, by Definition 4, equivalent to $\neg P \rightarrow Q'$, this last formula is equivalent to

$\exists Q_1(\models(Q \rightarrow [(R_p \circ R_b)^*]Q_1) \wedge \models((Q_1 \wedge \neg P) \rightarrow Q'))$.

Let $Q'' = [(R_p \circ R_b)^*]Q_1$ where Q_1 is a predicate whose existence is guaranteed by the previous formula. Then by definition of Q'', $\models(Q \rightarrow Q'')$. By Lemma DI (c), $\models(Q'' \rightarrow Q_1)$. This fact and $\models((Q_1 \wedge \neg P) \rightarrow Q')$, imply that $\models((Q'' \wedge \neg P) \rightarrow Q')$. Thus we need only show that $\models((Q'' \wedge P) \rightarrow [R_b]Q'')$. By Lemma DI (d), $\models(Q'' \rightarrow [R_p \circ R_b]Q'')$, which by Lemma DI (b) and Definition 4 implies $\models((Q'' \wedge P) \rightarrow [R_b]Q'')$.∎(Lemma D2).

Lemma D3: $(Q, Q') \notin \mathcal{R}_{while\ p\ do\ b}$ implies

$\neg\exists Q''(\models((Q'' \wedge P) \rightarrow [R_b]Q'') \wedge \models(Q \rightarrow Q'') \wedge \models((Q'' \wedge \neg P) \rightarrow Q'))$.

Proof of D3: If $(Q, Q') \notin \mathcal{R}_{while\ p\ do\ b}$ then by Definition 2, there exists s,s' satisfying

(†) $\quad Q(s) \wedge (s,s') \in R_{while\ p\ do\ b} \wedge \neg Q'(s')$.

Since $(s, s') \in R_{while\ p\ do\ b}$, we have $\neg P(s')$, and, as we observed in Appendix A, there is a sequence of states $s_0, \ldots, s_k$ such that

(i) $s_0 = s$,

(ii) $s_k = s'$, and

(iii) $P(s_i) \wedge (s_i, s_{i+1}) \in R_b$ for all nonnegative integers $i<k$.

Now assume Q'' satisfies

(iv) $\models(Q \rightarrow Q'')$,

(v) $\models((Q'' \wedge P) \rightarrow [R_b]Q'')$, and

(vi) $\models((Q'' \wedge \neg P) \rightarrow Q')$.

If k=0, then $s=s'$, so $\neg P(s)$, (iv), and (vi) together imply $Q(s) \rightarrow Q'(s')$, which contradicts (†).

If k>0, then by (†) and (iv) we have $Q''(s_0)$. By (iii) and (v) we conclude $Q''(s_i)$ for $i \leq k$. Then $\neg P(s')$, (ii), and (vi) imply $Q'(s')$, again contradicting (†).∎(Lemma D3).

We can now complete the proof of Theorem 5.

Suppose a = *while* p *do* b. Then $(Q, Q') \in \mathcal{M}_a$

iff (by PC4) $\exists Q''[((P \wedge Q''), Q'') \in \mathcal{M}_b \wedge \models(Q \rightarrow Q'') \wedge \models((Q'' \wedge \neg P) \rightarrow Q')]$

iff (by induction) $\exists Q''[((P \wedge Q''), Q'') \in \mathcal{R}_b \wedge \models(Q \rightarrow Q'') \wedge \models((Q'' \wedge \neg P) \rightarrow Q')]$

iff $\exists Q''[\models((P \wedge Q'') \rightarrow [R_b]Q'') \wedge \models(Q \rightarrow Q'') \wedge \models((Q'' \wedge \neg P) \rightarrow Q')]$

iff (by Lemmas D1 and D2) $(Q, Q') \in \mathcal{R}_a$. ∎(Theorem 5)

Appendix E.

Proofs of Theorem 12, Corollary 13, and Theorem 14.

Theorem 12: Let P be a predicate true in only finitely many states. Then $P\langle a\rangle Q$ is derivable from TA1-7 *iff* $P\langle a\rangle Q$ is true for the standard relational semantics.

In what follows, truth of termination assertions will be counted with respect to $\mathbf{R}$, and derivability with respect to TA1-7.

Soundness of the system TA1-7 follows directly from the definitions. That is, if $Q\langle a\rangle Q'$ is derivable, then it is true. We omit details of the proof.

Conversely, we prove completeness for termination assertions with finite antecedents by induction on program structure. Namely, if $Q\langle a\rangle Q'$ is true and Q is finite, then $Q\langle a\rangle Q'$ is derivable.

If a is *NOP* or a primitive statement, then all true assertions $Q\langle a\rangle Q'$ are obviously derivable from TA1, 2 and 6.

If a is $b;c$ and $Q\langle b;c\rangle Q'$ is true, then by TR2 and Definition 6 of termination assertions, there is, for every state s such that $Q(s)$, a state sequence $s...s''...s'$ such that $Q'(s')$, $s...s' \in Tr_b$, and $s''...s' \in Tr_c$. Let $Q'' = \vee\{E_{s''} \mid Q(s)\}$. Note that if Q is finite, so is Q''. Moreover, $Q\langle b\rangle Q''$ and $Q''\langle c\rangle Q$ are true, and so by induction hypothesis they are derivable. But then $Q\langle b;c\rangle Q'$ is derivable from them by TA4.

If a is *while* p *do* b, we first observe that if $E_s\langle a\rangle E_t$ is true, then it is derivable. For if $E_s\langle a\rangle E_t$ is true, then as in the proof of Lemma A1 we have either

(i) $dist_{b,P}(s,t) = 0$, so $s = t$ and $\neg P(s)$, and therefore $E_s\langle a\rangle E_t$ is derivable by TA3, or

(ii) $dist_{b,P}(s,t) = n+1$, $E_s\langle b\rangle E_{s_1}$ is true, $E_{s_1}\langle a\rangle E_t$ is true and $dist_{b,P}(s_1,t) = n$, for some state s_1. Then by induction on n, $E_{s_1}\langle a\rangle E_t$ is derivable, and by the main induction hypothesis $E_s\langle b\rangle E_{s_1}$ is derivable, and therefore $(P\wedge E_s\langle a\rangle E_t \wedge\neg P)$ is derivable from TA5. But since $dist_{b,P}(s,t) > 0$, we have $P(s)$, and hence $P \wedge E_s = E_s$. Also since $E_{s_1}\langle a\rangle E_t$ is true, we have $\neg P(t)$ and hence $E_t \wedge \neg P = E_t$. Therefore $E_s\langle a\rangle E_t$ is derivable.

Finally, if $Q\langle a\rangle Q'$ is true, then it follows by definition that for every s such that $Q(s)$, there is a t such that $Q'(t)$ and $E_s\langle a\rangle E_t$ is true. By the preceding observation $E_s\langle a\rangle E_t$ is derivable, and hence $E_s\langle a\rangle Q'$ is derivable by TA6. But $Q = \vee\{E_s \mid Q(s)\}$, so if Q is finite, then $Q\langle a\rangle Q'$ is derivable by TA7 from $\{E_s\langle a\rangle Q' \mid Q(s)\}$.∎(Theorem 12).

Corollary 13: $\langle R_a \rangle Q = \vee\{P \mid P(a)Q \in Th(TA1\text{-}7)\}$.

Proof: $(\langle R_a \rangle Q)(s)$ *iff* (by definition of $\langle R_a \rangle$) $\exists t[(s,t) \in R_a \wedge Q(t)]$

iff (by Definition 6) $E_s(a)Q$ is true

iff (by Theorem 12) $E_s(a)Q$ is derivable

iff (by TA6) $\exists P[P \wedge E_s = E_s$ *and* $P(a)Q$ is derivable]

iff the righthand predicate is true of s.∎(Corollary 13).

Theorem 14: The termination assertions derivable from TA1-8 are precisely those true for the standard relational semantics.

Proof: Soundness of TA8 follows immediately from the logical equivalence of $\forall i[P_i \rightarrow Q]$ and $\exists i[P_i] \rightarrow Q$.

To prove completeness, assume $P(a)Q$ is true. Then $E_s(a)Q$ is true for all s such that $P(s)$, and so is derivable by Theorem 12. But since $P = \vee\{E_s \mid P(s)\}$, we can derive $P(a)Q$ by TA8.∎(Theorem 14).

intuitive reasoning indicated before Lemma 7. Namely, the pre-condition which "will certainly

Notes

1. In a later paper Hoare [1978] emphasizes the verification aspects of proof rules rather than their use as a means of specifying semantics. However, it is the latter use on which we focus here.

2. We avoid the use of the word "nondeterministic" here because nondeterminism is a property of how final states *are computed* from initial states, rather than merely being a property of which initial states *map* to which final states. Nonfunctional relations can only arise from nondeterministic programs, but functional relations do not necessarily arise only from deterministic programs.

3. We emphasize that in this paper predicates are mathematical objects, not to be confused with expressions denoting them.

4. This definition is essentially taken from Pratt's [1976, Section 3.1, p. 115] definition of "ω-ary relation semantics."

5. Technically speaking, HL1-5 are axiom schemes in which a, b may be any programs, Q any predicate, etc.

6. Pca's are usually defined to be wholly syntactic objects, namely, to be of the form p{a}q where p, q are predicate *expressions*. Similarly, in the thesis on which the Hoare and Lauer paper is based, Lauer [1971, p. 67] defines a syntactic version of partial correctness semantics. Namely, he associates with each program a certain set of pairs of predicate expressions.

However none of the theory we treat in this paper depends on the syntax of the expressions used for predicates, and for this reason it is simpler to let the predicates themselves appear as components of pca's. In this way we avoid having to introduce rules for syntactic manipulation of expressions, and also avoid problems arising from the fact that certain predicates may not be definable using a particular class of expressions. We plan to treat these latter problems in a later paper (*cf.* Section 6, third paragraph).

7. A similar observation is made by Pratt [1976, section 1.2].

8. Indeed, partial correctness semantics are potentially richer than relational semantics because not all partial correctness semantics are derived from relational semantics. Put formally, any relation, $\mathcal{M}$, on predicates must be contained in $\{(P, Q) \mid (P, Q) \text{ holds for } max(\mathcal{M})\}$, but the containment is proper in general. Such an $\mathcal{M}$ does not correspond to the set of all partial correctness assertions true for any single relation on states. We shall, however, make no use in this paper of partial correctness semantics which are not equivalent to relational semantics.

9. The "box" notation is taken from the "necessity" operator of modal logic following the

respect to the other states. With some care, an

inductively with the result that the extended initial-state, final-state relation of a program does indeed determine its wp (*cf.* [deBakker, 1978]). However, the converse difficulty, that wp_a does not determine either R_a or $R_a^{\perp}$ remains.

suggestion of Pratt [1976]. Dijkstra [1976] refers to weakest antecedents as *weakest liberal preconditions*. Schwarz [1974] called them *exact semantic backward connections*.

10. Hoare and Lauer omit FH1 -- a minor oversight. Their D3 is misprinted, but it is clear from their Lemma 9 that they intended to state FH4.

11. In Lauer's thesis [1971] a "rule of consequence" equivalent to FH5 is included.

12. Theorem 5, in particular the characterization of *while* statements by PC4, is implicit in Lemma 2.3 of deBakker [1975].

13. Technically speaking, we should say that $\mathbf{R}$ is a model of Th(FH1-5) and Th(T1-5) where Th($\mathcal{S}$) refers to the set of theorems deducible in deductive system $\mathcal{S}$.

14. This seems to be the technical content of the frequently heard remark that pca's cannot be used to *assert* termination, (*cf.* e.g. [Hoare, 1969], [Manna, 1974], [Pratt, 1976]). The remark is correct, but must not be misinterpreted as implying that pca's are an inherently inadequate means of specifying semantics. As we have seen in Lemma 2, the complete set of true pca's uniquely determines $\mathbf{R}$ despite the anti-monotonicity of pca's.

15. Hoare and Lauer refer to "theorems...proved in the relational theory", rather than mentioning models explicitly. They do not offer rules of inference for proving theorems from HL1-5, although Lauer [1971] indicates by example that he intends the usual rules of predicate calculus to be applied. We prefer the more general definition of consistency (implication) we have formulated in terms of models, since this definition is not vulnerable to failures springing from the frequently unavoidable incompleteness of effective inference rules. If we assume that a complete set of inference rules are available, or alternatively if we use the model-theoretic notion of theorem, namely, an assertion is a theorem when it is true of all models of the axioms, then their formulation of consistency is equivalent to ours.

16. Note that although FH1-4 specify $\mathbf{R}$ according to the technical hypotheses of Theorem 9, it is not true that Th(FH1-4) = Th(FH1-5), or even that $\mathbf{R}$ is the largest model of Th(FH1-4).

We regard the characterization of $\mathbf{R}$ in Theorem 10 as subtle because there is no particular reason to look at largest models in the context of an axiomatization like HL1-5. Indeed, we saw that by adding HL6-7 only one model is possible, so there is no reason to expect or rely on a condition like maximality to force uniqueness.

17. A similar observation is made by Raulefs [1977, Lemma 2-1, p. 11].

18. Dijkstra's definitions can be found on pp. 25, 30 and 35. Our WP3 can be obtained from Dijkstra's definition of the **do...od** construct (p.35) by noting that *while* p *do a* is equal to **do** p → *a* **od**.

19. Note that for programs which are not primitive statements, the restriction that R_a be a function is not sufficient to ensure $\langle R_a \rangle = wp_a$ (*cf.* the examples in section 5.4).

20. Lemma 7 does not generalize much beyond the trivial class of *while* programs we are considering. In particular, when nondeterministic program constructs such as guarded commands are allowed in programs, Lemma 7 does not hold.

21. Recapitulating the developments in earlier sections, we could regard TAI-7 or TAI-8 as specifying the standard relational models rather than possible termination transformers. Indeed Theorem 7 together with Theorems 12 and 14 immediately imply that $\mathbf{R}$ is the *smallest* relational model of TAI-7 and/or TAI-8. Since termination assertions, like transition assertions, are monotone, we cannot hope to determine $\mathbf{R}$ as the *unique* model of TAI-8.

However, note that by fashioning a deductive system involving both pca's and termination assertions, using FHI-5 and TAI-7 for example, one *can* obtain $\mathbf{R}$ as the unique model. Our point here is that there is no special inadequacy of deductive specifications which prevents them from determining a unique model.

22. Alternatively, we can reach the same definition of wp_{A_3} by regarding A_3 as the nondeterministic join of A_1 and A_2. That is, using Dijkstra's guarded command notation, A_3 is equivalent to if $true \rightarrow A_1 \parallel true \rightarrow A_2$ fi, so that $wp_{A_3} = wp_{A_1} \wedge wp_{A_2}$ [Dijkstra, 1976, p.34].

March, 1979

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office/Boston
495 Summer Street
Boston, MA 02210
1 copy

Office of Naval Research
Branch Office/Chicago
536 South Clark Street
Chicago, IL 60605
1 copy

Office of Naval Research
Branch Office/Pasadena
1030 East Green Street
Pasadena, CA 91106
1 copy

New York Area Office
715 Broadway - 5th floor
New York, N. Y. 10003
1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375
6 copies

Assistant Chief for Technology
Office of Naval Research
Code 200
Arlington, VA 22217
1 copy

Office of Naval Research
Code 455
Arlington, VA 22217
1 copy

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380
1 copy

Office of Naval Research
Code 458
Arlington, VA 22217
1 copy

Naval Electronics Lab Center
Advanced Software Technology
Division - Code 5200
San Diego, CA 92152
1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Math Department
Bethesda, MD 20084
1 copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
(OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York, N. Y. 09501
1 copy